Honors Thesis: An Empirical Approach to Models of Associative Learning for Sequence Memory

Nathan Hayes, Advised by Dr. James Reggia
For the University of Maryland Department of Computer Science

April 2022

Abstract

Work of Sylvester et al. [1], Winder et al. [2], Horn and Usher [3] has established the potential of associative learning models to perform sequential memory tasks. They have accomplished this by combining standard Hebbian learning with a component which transitions between activity states. While their work focuses on these models as reflective of human sequential memory, this work examines them as machine learning tools and modifies them with the goal of improving recall. Specifically, it addresses the model of Sylvester et al. [1]. An alteration of the model is produced primarily by restructuring use of the θ parameter, and the resultant models demonstrate improved sequential recall. These improvements are examined in both the orthogonal and non-orthogonal context and compared.

1 Introduction

One substantial motivation for associative and Hebbian learning research lies in the potential for models such as Hopfield networks to emulate human neural activity [4]. One facet of this is the desire to mimic human sequential memory, namely, our ability to store sequences and patterns in our working memory and recall them later. Evidence suggests that this form of memory functions fundamentally differently from non-sequential memory in that adjacent items in a pattern are somehow linked in memory; remembering something in sequence prompts recall of subsequent items [5]. Furthermore, there is a recency bias of memory in that humans are more likely to remember later elements in patterns [2]. These traits have motivated research into modifications of associative learning models to capture this unique behavior.

Work of Horn and Usher [3] forms the foundation of one such model. Specifically we consider work of Sylvester et al. [1] and Winder et al. [2] which expands on this model. These papers introduce a 'transition matrix' V to the standard Hopfield network structure, and modulate its use by a parameter θ which prevents the network from resting in a minimum for too long. The primary goal of this paper is to investigate the behavior of this model in order to understand ways it may be optimized for use as a machine learning tool. Potential applications for such a model which has robust recall are numerous, and include storage of large pieces of data for recall in a memory efficient manner, as well as more straightforward applications to sequential memory.

To accomplish this goal, we first investigate the model of Sylvester et al. thoroughly to determine its characteristics. Once this is done, we identify areas in which recall might be improved, propose modifications which we believe will be beneficial, and empirically evaluate their value by implementing them and comparing their recall to the original model. We then discuss the ways in which

our modifications succeeded and failed, propose explanations for the results, and evaluate the practical merits of the resultant models. In particular, we focus on the disparity between storing orthogonal patterns and non-orthogonal patterns in these models, as this is the primary roadblock to practical use of many Hebbian learning models.

2 Background

2.1 Orthogonalization

In the context of Hopfield networks and Hebbian learning, the ability to orthogonalize or approximately orthogonalize input patterns is crucial for accurate recall and storage capacity. As such, many different techniques have been attempted to accomplish this. Some such techniques employ known orthogonalization algorithms, including the Gram-Schmidt method [6][4][7][8][9][10] and symmetric orthogonalization due to Löwdin [6]. Of these, the frontrunner seems to be Gram-Schmidt [6][9], for a couple of reasons. First, since it is a hierarchical method, it is easy to orthogonalize new vectors coming in rather than reprocess the whole set stored in memory, and so it is more time-efficient than its alternatives. Second, the structure of Hopfield networks is such that it is mathematically simple to orthogonalize inputs, since one need only subtract the field generated when inputting a new pattern to do so. However, this technique has its own flaw, which lies in its time consumption. Since this involves a full matrix multiplication, it has $O(n^2)$ time complexity to learn each individual pattern, leading to a total $O(mn^2)$ time complexity to store m n-bit vectors. Furthermore, while Gram-Schmidt will maintain each data point as an attractor, it often fails to maintain a large basin of attraction around these points. Empirically, with loads of around $0.24N$, Gram-Schmidt Hopfield nets will re-

turn stored patterns when fed in exactly those patterns. However, when data was slightly corrupted, only around 50% of trials returned the original pattern successfully, indicating a large number of spurious attractors [8].

One simple way to alleviate the time-cost issues of Gram-Schmidt and other orthogonalization methods is to perform 'pseudo-orthogonalization', where instead of finding truly orthogonal vectors, the data is scrambled in some reversible way so that the expected correlation between two data points is close to 0. The simplest way to do this is to generate a random vector in $\{-1, 1\}^n$, and then just element-wise multiply our existing data point with this vector [11]. Multiplying by the same vector again recovers our original pattern. Alternatively, some methods add additional neurons [12] and augment input patterns for the express purpose of orthogonalization, although methods for determining how to augment the inputs appropriately have mixed success.

More sophisticated methods involve using complex or quaternion valued Hopfield nets, and masking our original pattern within a partially random complex or quaternion value [11]. These methods are generally more effective, but involve larger Hopfield nets, as using complex numbers for example requires a Hopfield network of size $2N$, for data points of size N. The benefit to using these methods is that they have much faster operation times, which can be further shortened by using the same random values for fixed-size blocks within each data point, meaning that with blocks of size k, you only need random vectors of size N/k. However, these methods require external storage of a large number of random patterns, and are generally less effective than true orthogonalization.

Expansions and generalizations of Hebb's learning rule have been formulated to try to overcome the stability problems inherent in Hebbian learning. One of the first and most influential such modifications is Oja's rule, which performs a first order approximation of normalization of weight changes to prevent weights

from growing unboundedly. Sanger further improved on this by developing the Generalized Hebbian Algorithm, which incorporates the Gram-Schmidt process into Oja's rule [9]. Both techniques are approximate, and so there is a trade-off between learning rate and likelihood of convergence, but empirical results find them to be quite effective. These techniques are also related to principal component analysis, as we briefly touch on later.

A more recent method uses a matrix known as a conceptor. Given a set of data points, conceptors try to act as the identity matrix for our data points while being small, with the result being that they are linear objects which identify the most relevant dimensions for any given data set. Importantly, conceptors from two different data sets can be easily appended together, and 'conjugate' conceptors that identify 'unused' dimensions can be easily identified. This allows us to identify the most relevant dimensions for a set of data, and then project further data points onto remaining dimensions. Unlike Gram-Schimdt, this allows data to span multiple dimensions while still maintaining orthogonality between the groups of data. This approach is fairly new, but when applied to simple feedforward nets trained to categorize images, conceptor-aided backpropagation performed as well as the best known previous methods, while using fewer layers and nodes [13].

For data that is initially biased, traditional Hopfield nets generally perform very poorly, lacking the ability to even maintain the stored patterns as local minima. Applying gram-schmidt orthogonalization helps somewhat by ensuring that each pattern is at least a fixed point, but with biased data Gram-Schmidt produces a large number of spurious attractors, to the point where many stored patterns have a basin of attraction of size 0, making it not much better in practice. To better resolve this issue, a global hard or soft constraint can be introduced, which either forces the Hopfield net to maintain certain bias ratios

or makes it more easy to flip to whichever data point the data is biased in favor of. Interestingly, this adjusted algorithm has a better storage capacity than traditional Hopfield nets, which increases with bias to load factor of around 0.18 for large values of bias. This implies that generating biased patterns and then building a Hopfield net that 'corrects' for this bias may lead to more effective storage than even with unbiased patterns [14].

It is interesting to think about Hebbian learning as a form of principal components analysis. Traditionally, PCA identifies dimensions along which a set of data has the greatest variation, which allows you to project data onto decorrelated vectors. Hebbian learning does something similar, with a weight matrix that changes depending on the correlations between neurons. In this sense, the weight matrix's eigenvalues can be seen to be the principal components that are extracted, with the subtraction term in the Hebbian learning rule being exactly the right term to prevent the matrix from growing without bound [15].

2.2 Oscillatory Hebbian Networks and Sequential Memory

The study of oscillatory Hebbian networks has developed rapidly in the last few decades, as a means of connecting relating real world behavior of human brains to mathematical models. Early empirical discoveries relevant to this subfield were made by Eckhorn et al. [16], who observed periodic firing of neurons in the visual cortex. Citing this observation, Horn and Usher [3] developed a modified Hebbian network, which dynamically reduced the threshold needed to move to another state, which prevented convergence to a single steady state. This innovation allowed their network to generate oscillatory behavior, which allowed them to store a superposition of patterns in the form of oscillating 'final states.'

The study of sequential memory builds on this body of work by introducing the idea of time asymmetry. Notably, it is well known that human memory exhibits a 'recency' bias, meaning that when learning a sequence of patterns, we tend to better remember later patterns. This observation was incorporated by Winder et al. into Horn and Usher's model through the use of a time biased Hebbian learning rule [2]. Mathematically, this means that wherever a new pattern was learned, the weight of all previous patterns was scaled down by a constant factor, resulting in stronger weights for later patterns. With non-0 decay factors, this model successfully increased the rate of recall for later patterns, at the cost of recall for earlier patterns. Notably however, while this model both oscillates and biases towards remembering later patterns, it does not oscillate in any specified order, which is somewhat contrary to the idea that patterns are learned sequentially.

Addressing this, Sylvester et al. [1] introduce a new component V to the model, which pushes the model to pattern t when at pattern $t-1$. Mathematically, the matrix V is constructed via incremental learning rule:

$$v_{ij}^{t} = (1 - k_{dv})v_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^{t-1}$$

By adding V into the existing model with dynamic thresholds, they managed to successfully recall patterns in the order in which they were stored. Our work will focus heavily on modifications of this idea. In particular, the theoretical behavior of this V matrix, for both orthogonal and non-orthogonal stored patterns, is directly relevant to our work, and so is explored in greater depth in Section 4.2.

3 Observations of the Model

We began our work by implementing the model described in Sylvester et al. [1] and observing its operation. We constructed plots of its behavior and the patterns it produced and discovered a curious phenomenon. The model consistently falls into repeating cycles when allowed to run for sufficient time. This trend we expected, but what surprised us was the nature of these patterns - specifically, we realized that the model consistently would produce sequences of patterns closely matching those stored, hit the last pattern in the sequence, and then go blank for a period of time. This would repeat ad infinitum. These blank periods, in which the model's output was not close to any stored pattern, were unusually consistent in that we observed them no matter what patterns we stored in the network. With a little work, we discovered the cause - negatives. Whenever a pattern is stored in this type of network, it's negation is equivalently stored. During the blank periods, it was these negative patterns which were being produced.

Upon modifying our plots to account for these negated patterns, we were able to see the behavior more clearly, and we hypothesized that the nature of the θ parameter was the cause. When the final pattern in the sequence settled, θ would eventually flip most of the bits and result in a negated pattern. Once we had made this observation, we conjectured that we could improve on the results of the model by changing the direction θ pushed, and thus prevent it from flipping. In addition, we observed behavior in which a pattern is partially reached and then passed over, indicating that the model did not smoothly transition between stored patterns. We hypothesized that the nature of θ caused this interference. Since θ acts independently on each bit, we theorized that it would force certain bits out of their fixed points before the convergence of others. In order to address these concerns regarding our observations of the

model's behavior, we proposed two simple modifications to the model. We were interested to see if these modifications would resolve the concerns raised. In addition, we wanted to test whether they would interact with orthogonal patterns in a different way than with random patterns. We present both the analysis we have done and the evidence we have gathered to test our hypotheses.

4 Methods

4.1 Model Description

We will consider three distinct Hopfield-network models, distinguished by different learning rules. Each model uses a fully connected set of N linear threshold units, which take values from $a_i = \{-1, 1\}$. We denote a_t to be the full state of all N nodes at time t, and refer to node i's state by $a_{t,i}$. Given a sequence of input patterns, $\{a_p\}_{p\in\{1,\dots,M\}}$, we generate two matrices W, V via the following formulas:

$$W = \frac{1}{N}\sum_{p=1}^{M}(1-k_{dw})^{M-p}(a_p a_p^T - I)$$

$$V = \frac{1}{N}\sum_{p=2}^{M}(1-k_{dv})^{M-p}(a_p a_{p-1}^T)$$

This is equivalent to the definition of W and V given in Sylvester et al. [1], which updates W, V as each successive input is read. These two matrices serve as the building blocks for our three weight matrices. Conceptually, W is a standard Hopfield network weights matrix, which tries to make each input pattern a steady state. V on the other hand induces a temporal ordering on the patterns, with the goal being to move the model from a_p to a_{p+1}.

Alongside these two matrices, Sylvester et al. introduce a third component, which is a time-varying threshold for each node i, denoted θ_i^t. The vector of all

such thresholds is denoted by θ_i. At each time-step, their model then updates every node simultaneously, via:

$$a_{t+1} = sign(\beta_1 W a_t + \beta_2 V a_{t-1} - \theta_t)$$

β_1, β_2 are constants that determine the relative weights of W and V. In the edge-case where $\beta_1 W a_t + \beta_2 V a_{t-1} - \theta_t$ is exactly 0 in some dimension i, the model defaults to not changing node i's state. Firing simultaneously has the negative effect of allowing the model to alternate between two patterns that are not steady states. However, as discussed in the next section, it also has theoretical benefits. θ_t itself updates each time-step, by increasing if node i has not changed, and decaying if it has. Specifically, given decay and growth rates $0 < k_\theta < k_w < 1$, θ_i^t updates via

$$\theta_i^{t+1} = (1 - k_\theta)\theta_i^t + k_w \cdot |a_{t,i} - a_{t-1,i}|$$

Thus, nodes should not remain in any given state indefinitely, as the threshold will decrease until it is forced to switch signs. Conceptually, the Sylvester model relies on θ to force the model to alternate, V to push the model from one pattern to the next, and W to induce convergence to a steady state after the initial push.

4.2 Theoretical Analysis

Our focus in this work is finding alternative methods of forcing the model to leave a steady state, which we refer to as 'destabilization'. To accomplish this, we perform an analysis similar to that of Winder et al. [2], focusing on the inputs generated by our stored patterns, before then considering the impact of various alternative destabilization methods. Following their notation, we define

$h_V^q := Va_q$, and $h_W^q := Wa_q$, representing the input to the network at each of the stored patterns. We can then compute:

$$h_W^q = [(1-k_{dw})^{M-q} - \frac{1}{N}\sum_{p=1}^{M}(1-k_{dw})^{M-p}]a_q + \frac{1}{N}\sum_{p\neq q}^{M}(1-k_{dw})^{M-p}a_p a_p^T a_q$$

$$h_V^q = Va_q = \frac{1}{N}[(1-k_{dv})^{M-q-1}(a_{q+1}a_q^T a_q) + \sum_{p=2,p\neq q+1}^{M}(1-k_{dv})^{M-p}(a_p a_{p-1}^T a_q)]$$

We first consider the case of orthogonal patterns, for which $a_q \cdot a_p = 0$. This then all simplifies to:

$$h_W^q = [(1-k_{dw})^{M-q} - \frac{1}{N}\sum_{p=1}^{M}(1-k_{dw})^{M-p}]a_q$$

$$h_V^q = Va_q = (1-k_{dv})^{M-q-1}(a_{q+1})$$

In this simple case, it is clear that W tries to make a_q a fixed point, whereas once at a_q, V tries to push the network to the next state. Importantly, because V only pushes to the next state when at a_q, we follow Sylvester et al. [1] in using simultaneous firing, to avoid the potentially ambiguous behavior that occurs halfway between patterns a_q and a_{q+1}. Moreover, when h_W^q overpowers h_V^q, the network stagnates near or at pattern a_q. To address this, the Sylvester model considers destabilization factor θ_t, which directly modifies the input via:

$$h^t = \beta_1 h_W^t + \beta_2 h_V^t - \theta_t$$

This has the desired effect of preventing the network from coming to rest. However, when the network stays at rest at pattern a_q, $-\theta_t$ gets closer each iteration to some multiple of $-a_q$. This pushes the network to the complementary pattern $-a_q$, which is always a spurious stored pattern, and explains why the

network tends to oscillate between patterns a_q and $-a_q$ under this formulation.

To address this, we noticed that when the network is stagnant at pattern a_q, the desired effect is to push the network specifically to the point a_{q+1}, which is the role of V. We have two methods of addressing this, one conservative and one aggressive.

4.3 Push V

In our first, more conservative approach, instead of appending θ to the input, we instead increase V relative to W. Formally, we define a new scalar variable θ' via:

$$\theta'_0 = 0$$

$$\theta'_{t+1} = (1 - k_\theta)\theta'_t + k_w \frac{bitdist(a_t, a_{t-1})}{N}$$

which modifies input h^t by:

$$h^t = \beta_1 h^t_W + \beta_2 \theta'_t h^t_V$$

We call this formulation 'Push-V,' because θ' increases the impact of V. Importantly, when the data are non-orthogonal, Va_q does not push the network directly to a_{q+1}, as there is interference from cross-talk terms. Therefore, it is important to also consider the case of non-orthogonal inputs. As in Winder et al. [2], we replace sums over vectors and dot products with their averages, which we denote by $a_u = \frac{1}{N}\sum_p a_p$ and $A_u = \frac{1}{N(N-1)}\sum_{p\neq q} a_p a_q$. We then obtain the following approximation from h^q_W from Winder et al.

$$h^q_W \approx \left[(1 - k_{dw})^{M-q} - \frac{1}{N}\sum_{p=1}^{M}(1 - k_{dw})^{M-p}\right] a_q + \frac{A_u}{N}\left[\sum_{p\neq q}^{M}(1 - k_{dw})^{M-p}\right] a_u$$

For V, we note that because we sum from $p = 2$ to M, these estimates have an additional source of error. Assuming that M is relatively large though, these errors can be assumed to be small, allowing us to derive an approximation for h_V^q:

$$h_V^q = \frac{1}{N}\Bigg[(1-k_{dv})^{M-q-1}(a_{q+1}a_q^T a_q) + \sum_{p=2,p\neq q+1}^{M}(1-k_{dv})^{M-p}(a_p a_{p-1}^T a_q)\Bigg]$$

$$\approx (1-k_{dv})^{M-q-1}(a_{q+1}) + \frac{A_u}{N}\Bigg[\sum_{p=2,p\neq q+1}^{M}(1-k_{dv})^{M-p}\Bigg]a_u$$

We observe that the leading term in both cases is identical to the case of orthogonal inputs, with the latter term being due to cross-talk, which pushes the network neither to a_q nor a_{q+1}. This error diminishes when the data becomes 'more orthogonal,' which is quantified by a decrease in A_u/N, for fixed N, M.

Under 'Push-V', the extent to which h^t pushes to a different pattern is approximately:

$$(\beta_1\Bigg[\sum_{p\neq q}^{M}(1-k_{dw})^{M-p}\Bigg] + \beta_2\theta'\Bigg[\sum_{p=2,p\neq q+1}^{M}(1-k_{dv})^{M-p}\Bigg])\frac{A_u}{N}a_u$$

Because $\theta' = 1$ produces the original model without destabilization, we should expect $\theta' > 1$ whenever destabilization is necessary. This, however, results in greater error due to cross-talk vs. the original model. Thus, for highly non-orthogonal values, we would expect that 'Push-V' decreases oscillation, but also results in worse retention. In the case of orthogonal or 'nearly-orthogonal' values, we hypothesize that this error should largely disappear, giving us better performance.

4.4 Snap-V

Our more aggressive approach does away with the balancing act between W and V entirely. Instead, the network will, by default, use only W, which should eventually bring the pattern to a fixed point. When such a state is reached, the network switches to entirely using V for one period, after which it reverts back to W. We refer to this approach as 'Snap-V,' due to the way the network snaps back and forth between using the two matrices.

This has major benefits in terms of error over our previous model. Consider when the network is stable at or around pattern a_q. For both Sylvester's model and Push-V, the effect of W is to preserve this state. However, when the network is storing pattern a_q, the goal of the network is to reach pattern a_{q+1}. This means that the effect of W should largely be seen as an error term. Similarly, when the pattern is falling into a_q, the goal of the model is to fully reach a_q, and so the effect of V is largely to confound the model. Conceptually, the goal of Snap-V is to eliminate this error by using W only when we want to fall into a state and using V only when we want to progress.

To achieve this, we need to define a metric m that measures when the network is falling into a pattern vs. seeking the next pattern. This is done by measuring the bit-distance b each step between the current and previous patterns, and adding $1/2^b$ to θ. When m reaches or exceeds 1, we switch to V, and reset m to 0. Formally:

$$m_t = \begin{cases} m_{t-1} + \frac{1}{2^{bitdist(a_t, a_{t-1})}} & m_t < 1 \\ 0 & m_t \geq 1 \end{cases}$$

When $a_t = a_{t-1}$, our model has successfully reached a stable state, and our metric reflects this by immediately adding 1 to m, triggering the 'snap.' Otherwise, our metric increments based on how close our two patterns are,

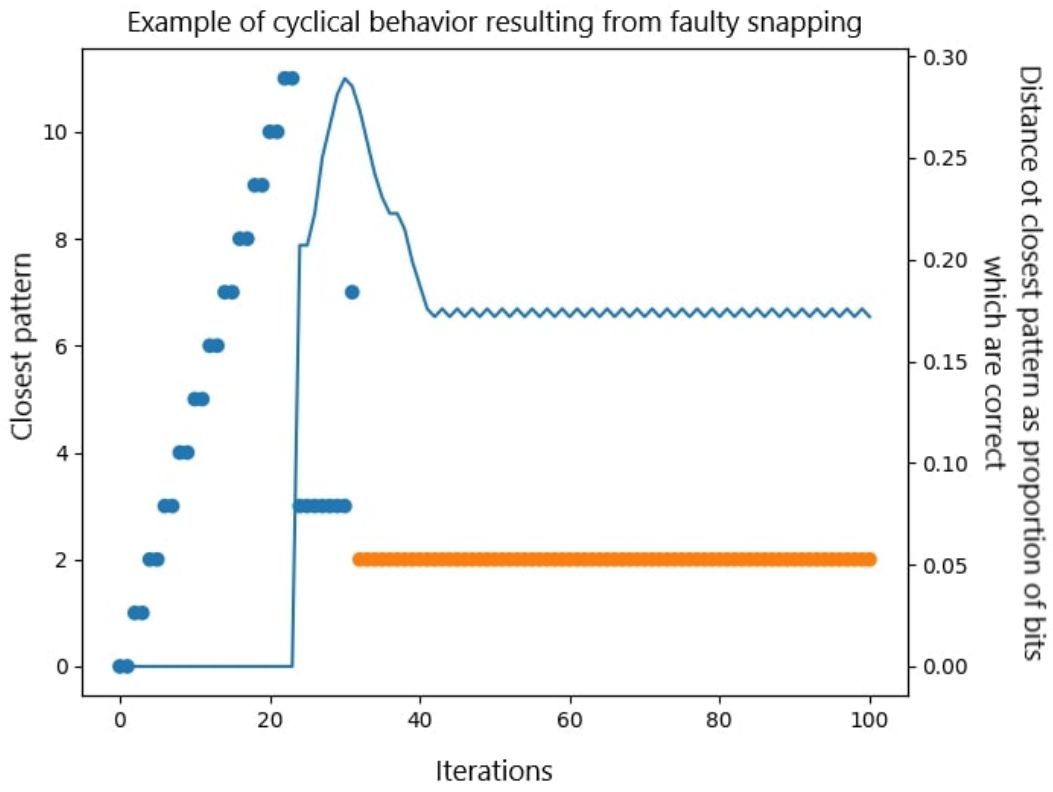

Figure 1: This a graph of the behavior of Snap-V when we only 'snap' if $a_t = a_{t-1}$. The points indicate which pattern we are closest to, measured along the left y axis. The line graph indicates the distance to the closest pattern, measured on the right y axis. We observe that the network cycles between 2 patterns ad infinitum, and thus never 'snaps'. This motivates our use of m.

which has the positive effect of avoiding stagnation in cyclical states, which is possible due to our use of simultaneous firing (see Figure 1).

4.5 Estimating Decay

In order to estimate optimal decay values for W and V, we again borrow from Winder et al. [2]. Specifically, after obtaining the approximation

$$h_W^q \approx \left[(1-k_{dw})^{M-q} - \frac{1}{N}\sum_{p=1}^{M}(1-k_{dw})^{M-p}\right]a_q + \frac{A_u}{N}\left[\sum_{p\neq q}^{M}(1-k_{dw})^{M-p}\right]a_u$$

Winder et al. argued that maximizing the ratio between the coefficient on a_q vs the coefficient on a_u should optimize for memory capacity. We should note that in their original paper, they nonetheless found learning rule decay rates through experimentation, and the approximation obtained via this theoretical method was 50% larger than optimal. However, due to computational limitations, we are unable to search in the same way, and so rely on this approximation to determine decay rates for both W and V. When using 30 patterns with net-size 256, we calculate optimal decay for W and V respectively to be 0.1007 and 0.132.

4.6 Experimental Methodology

To test our hypotheses, we began by considering the metric for success. First, we define the useful notion of a "chain." A *chain* is a sequence of model states such that if pattern A is recognized at time t_1 and pattern B is recognized at time $t_2 > t_1$, then B must come after A in our list of stored patterns. We say that the *length* of a chain is the number of distinct recognized patterns occurring in it.

After having observed the cycling behavior of the original model, we decided that it was diluting the original metric used in Sylvester et al., the average length of chains occuring in network behavior [1]. Since the network would cycle between a few states ad infinitum, the metric would only capture that cycling, rather than initial model behavior. As such, we decided to cut off the model once it had completed a chain. We also decided that we did not want to consider a chain to be valid if it skipped states. We refer to the resultant new

recall metric as s_{cut}.

It's important to note here that our consideration of complementary states plays into this decision. For ease of notation, if pattern A is stored in our network, we refer to its complementary pattern as $-A$. Per the original metric, a chain of states such as "-5 -6 -6 -7 -7 -8 -8 -9 -9", as shown in Figure 2 in iterations 26-34, would be counted as length 0, while we count it as length 5. In addition, the sequence "A B C -B C D" would be counted as a chain of length 4 (as -B would be ignored), but we decided that this should instead be a chain of 3, a chain of 1, and then a chain of 2, since we functionally treat negative patterns as distinct. We call this metric s_{cut}.

To accommodate this stringency we allowed a tolerance for the model state's distance to the desired pattern. For most of our work, this tolerance was 95%, although this parameter is easy to modify. This choice was the result of some experimentation. At low tolerances, low values of β_1 become optimal because W matters much less, but at high values the results become unreliable. Finally, we decided to also consider priming the model with its initial pattern in addition to random ones. This is primarily due to computational considerations. For random initialization, there is much more variance inherent to the results, and so we require many repetitions of our trials to be confident in our results. We still value the results of this method, but we supplement it with the primed method of initialization to bolster our conclusions.

Using this metric, we ran a cursory search of different parameters to our modified models to identify situations in which it performed well, both for random stored patterns and for orthogonal ones. In the process of doing so, we also ran this search on the original model to identify the ideal parameters. This resulted in our selection of models for analysis, consisting of those described in Table 1. We then directly compared on four tasks: primed recall for orthogo-

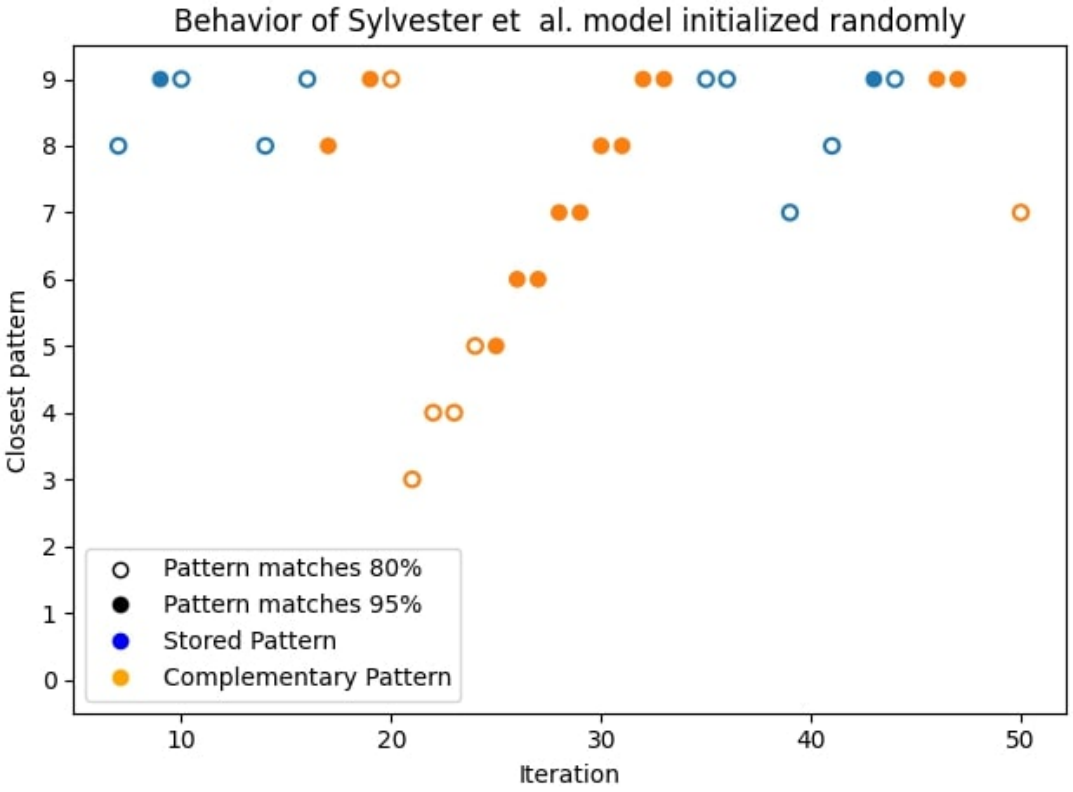

Figure 2: A chart of the states observed over a model's operation. The plot tells us which state the network is closest to at iteration t. Blue states are the patterns as originally stored, and orange states are the complements of these patterns. If the network is within 95% of its closest pattern, it is marked with a filled point. If within 80%, it is marked with a hollow point.

Model	k_θ	k_w	k_{dw}	k_{dv}	β_1	β_2
Sylvester et al. without decay	0.05	0.09	0	0	0.2	1
Sylvester et al. with decay	0.14	0.18	0.1007	0.132	0.7	1
Push-V without decay	0.20	0.40	0	0	0.2	1
Push-V with decay	0.20	0.36	0.1007	0.132	0.2	1
Snap-V without decay	N/A	N/A	0	0	N/A	N/A
Snap-V with decay	N/A	N/A	0.1007	0.132	N/A	N/A

Table 1: The parameters of our optimal models.

nal patterns, unprimed recall for orthogonal patterns, primed recall for random patterns, and unprimed recall for random patterns.

To perform this direct comparison, we generate sets of patterns to evaluate on. We used patterns of length 256 bits, and 30 patterns per learnset, well within the Hopfield learning capacity of $\sim 0.138N$. Non-orthogonal patterns are generated randomly, and the length of our patterns ensures that the minimum distance between patterns in our set is still quite large. Anecdotally, we observe that it is at least 20% of the length in the overwhelming majority of cases. For orthogonal patterns, we use a Hadamard construction, selecting random rows of the Hadamard matrix of size 256. We then train our models on these learnsets and evaluate their s_{cut} on each of the four tasks 200 times each. From this, we generate box plots of their recall across these iterations and draw our conclusions from the results.

5 Results

Our data definitively confirmed our hypothesis that both our modifications would prevent the oscillation behavior we observed. Note Figs. 3, 4, and 5. These have been selected for the clarity with which they display the effects we have observed, though the effects themselves are consistent across runs and independent of parameter choice or patterns stored.

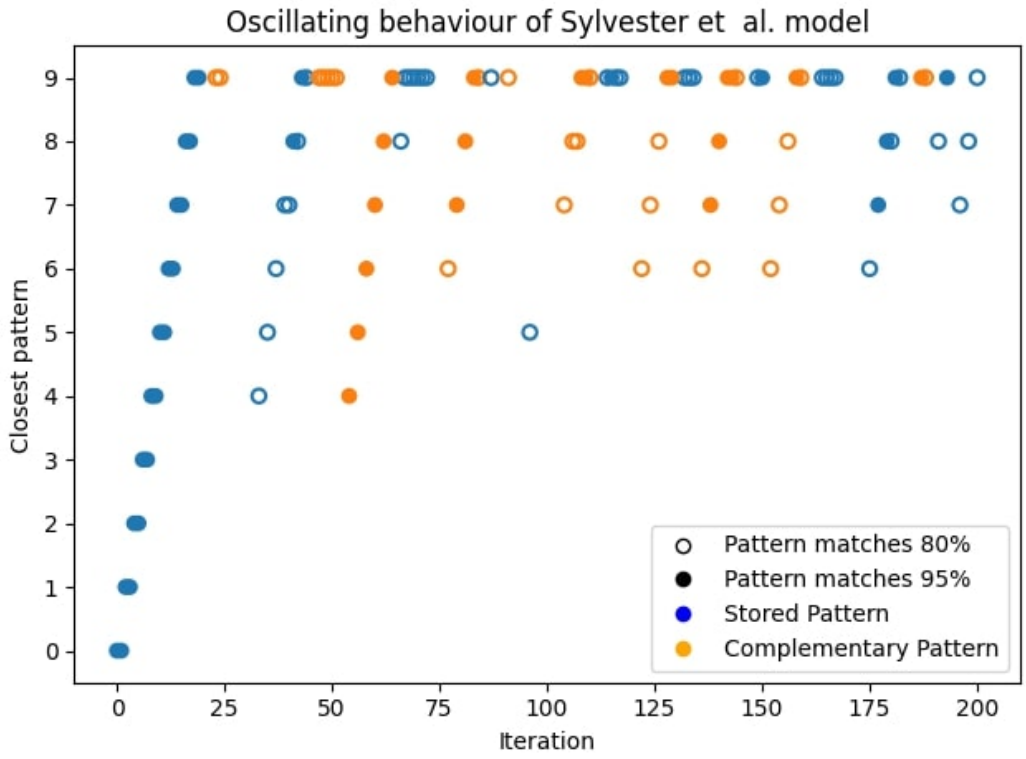

Figure 3: A state diagram of the operation of Sylvester et al.'s model, similar to Figure 2. Note the frequent switches between stored and complementary patterns.

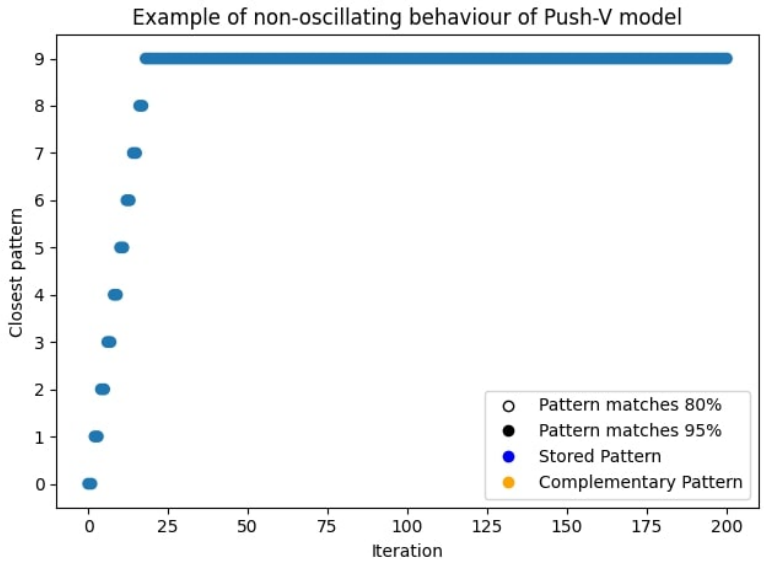

Figure 4: A state diagram of the operation of the Push-V model. Note the lack of switches between stored and complementary patterns.

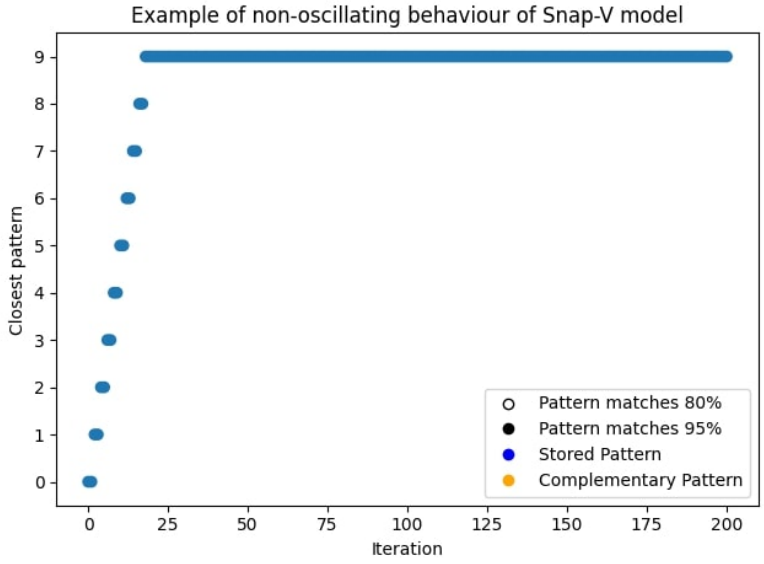

Figure 5: A state diagram of the operation of the Snap-V model. Note the lack of switches between stored and complementary patterns.

Orthogonal?	Seeded?	Using Decay?	Model Type	Mean	Std. Deviation
Y	Y	Y	Sylvester	1.00	0
			Push-V	30.0	0
			Snap-V	30.0	0
Y	Y	N	Sylvester	30.0	0
			Push-V	30.0	0
			Snap-V	30.0	0
Y	N	Y	Sylvester	1.47	0.782
			Push-V	1.55	0.960
			Snap-V	1.77	1.02
Y	N	N	Sylvester	6.56	5.66
			Push-V	8.97	6.00
			Snap-V	9.18	9.10
N	Y	Y	Sylvester	14.6	6.40
			Push-V	11.5	13.7
			Snap-V	24.9	9.01
N	Y	N	Sylvester	1.00	0
			Push-V	1.00	0
			Snap-V	1.00	0
N	N	Y	Sylvester	1.42	0.675
			Push-V	1.53	0.902
			Snap-V	1.93	1.07
N	N	N	Sylvester	0.46	1.95
			Push-V	1.33	3.69
			Snap-V	1.12	4.17

Table 2: The performance results of our optimal models.

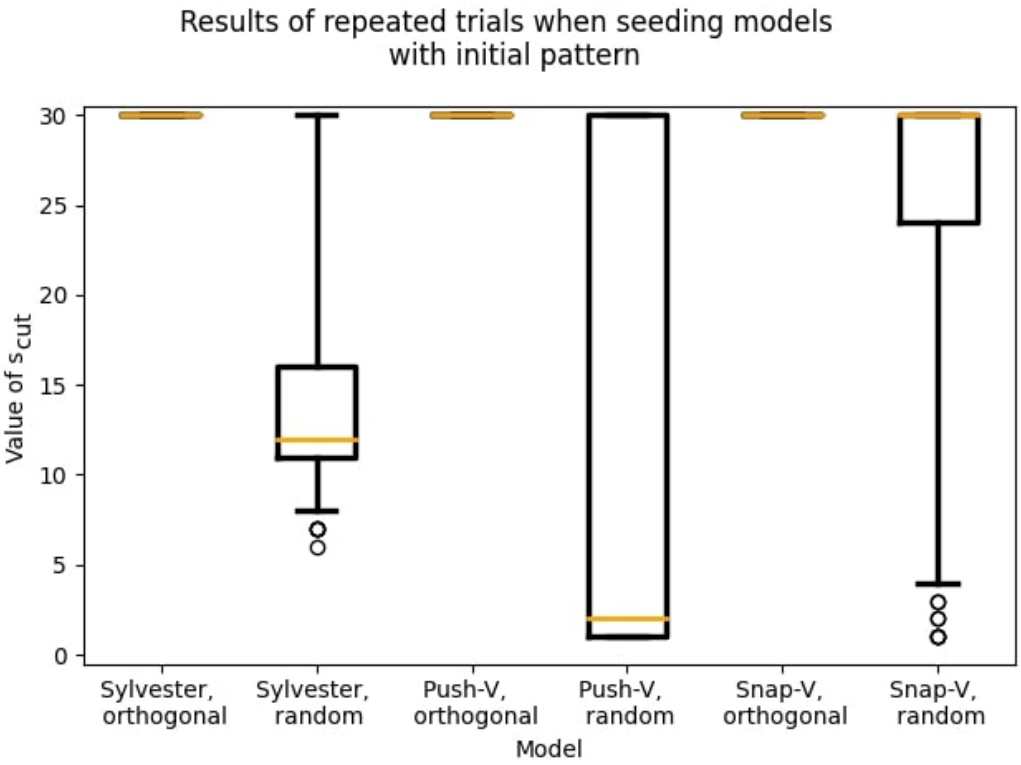

Figure 6: Distributions of the s_{cut} metric when 200 trials are run for each model. Here, models are initialized with the first pattern in the sequence. Note the perfect performance of each model on orthogonal patterns.

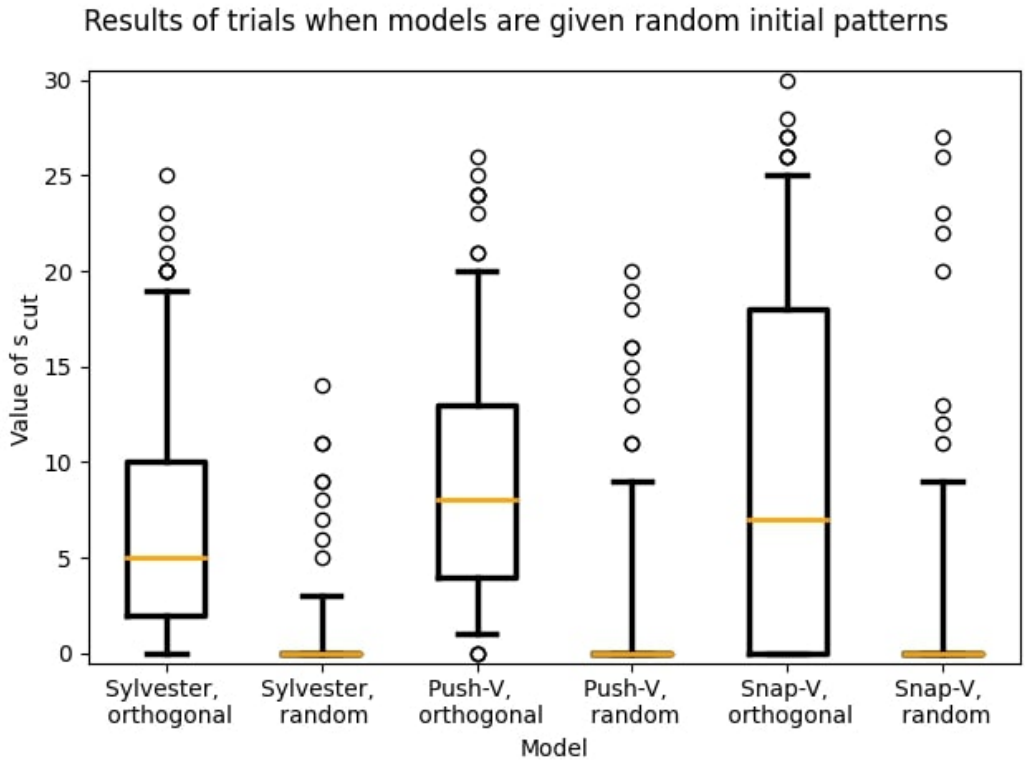

Figure 7: Distribution of the s_{cut} metric when 200 trials are run for each model. Here, models are initialized with a random pattern.

The optimal parameters identified by our search are listed in Table 1. In searching for the optimal parameters, one consistent and striking trend we noticed was that β_1 was often optimally quite small, meaning that the W matrix was unimportant to the model's behavior.

In terms of our tests of recall, the first and most important item to note is that in the vast majority of cases, models without decay (meaning 0 k_{dw} and k_{dv}) perform better than their counterparts which have positive decay. This effect is clear throughout most test cases, with the exception of the cases of orthogonal patterns seeded with the initial pattern, in which most models performed flawlessly. The notable and single exception to this is in Sylvester et al.'s model with decay, shown in Table 2. Potential reasons for this are discussed in Section 6.

We were successful in confirming our hypothesis that our Snap-V modification would improve the recall of the model, particularly in the case of non-orthogonal patterns when seeded with the initial pattern. In this case, it performs with comparable success to orthogonal patterns. To see this, note the significant improvements demonstrated in Figure 6. This is somewhat offset by a relative drop in performance in the randomly initialized case, as shown in Figure 7, although this seems to be a minor effect.

Unfortunately, we were unable to demonstrate any improvements on the part of the Push-V model. For many selections of parameters, both the original model and our modification performed flawlessly for orthogonal patterns when they were primed, which we believe to be a quality of both models which is maintained for any size of pattern. For non-orthogonal patterns, however, we observed that our modification did not significantly improve upon the recall of the original.

6 Discussion

While it is true that our models have eliminated the characteristic oscillatory behaviour of Sylvester et al.'s model, this is not a very practical consideration. In fact, this behavior is not directly detrimental to the function of the model, although it seems to be a symptom of problems endemic to the nature of θ in this model. These problems are discussed at length in Sections 3 and 4.2.

The optimal parameters discovered and listed in Table 1 are worthy of a great deal of discussion. First, it is important to note that the granularity of our search for these parameters was relatively low. This is due to practical considerations, as there is no obvious way to optimize choice of parameters besides exhaustive search, which is computationally intensive. As such, these should not be taken to be reliable or exhaustively-tested values. One worthy area of future research in this subject would be truly optimizing these parameters, either with more powerful hardware or more elegant techniques of optimization.

The parameters themselves are also worthy of note. In most cases, the optimal choice of β_1 was quite low, indicating that the W matrix was relatively unimportant to the successful operation of the model. It is difficult to fully explain this phenomenon, but we have several hypotheses. First, in our metric, pattern a_q was considered successfully reached if the current state a_t matches at least 95% of that pattern - in our test cases, this means at least 244 bits out of 256 matched. One possible explanation is that this threshold is overly lenient. Intuitively, once close to a pattern a_q, V has the tendency to 'slingshot' to the next pattern a_{q+1}. This means it is possible to get close to each pattern successively without ever needing W. With a more stringent metric, it is possible that the importance of W becomes more apparent. However, the relative success of the model in the random initialization contexts calls this explanation into question.

One notable effect of the granularity of our search lies in the decay variables. It was unfortunately impractical for us to search for optimal values for these variables. Instead, they were calculated from the normal operation of the model as discussed in Section 4.2. One consequence of this may have been the stark contrast between the success of models without decay and the relative failure of those with decay. It is possible that our changes to the metric, s_{cut}, also resulted in this effect. Unfortunately, without more exhaustive search, it is difficult to isolate the cause.

The failure of the Push-V model to significantly outperform Sylvester et al. seems to indicate that, although a useful stepping stone, Push-V does not go far enough in combating the error introduced by jointly applying W and V. This is most clearly indicated by comparison with Snap-V.

The most striking result of our work is in the Snap-V model, and crucially, its success on non-orthogonal patterns. This is most notable in Figure 6, where we see it outperform its equivalents for both Sylvester et al.'s model and the Push-V model. We believe this to be because it does not mix W and V, and thus minimizes error in contexts when only one is helpful. The success of Snap-V leads us to believe it may be promising for practical applications. This is because it seems to overcome the flaw of many Hebbian learning structures in that they require orthogonality to perform optimally, and in most situations orthogonal patterns are an unattainable luxury. With more robust stress-testing, it is our belief that this model may find use in large data or real-world sequential memory contexts.

7 Acknowledgement

This work was produced partially in collaboration with Yuang Shen, a student of University of Maryland in the Department of Mathematics.

References

[1] Jared C Sylvester, James A Reggia, Scott A Weems, and Michael F Bunting. A temporally asymmetric hebbian network for sequential working memory. In *Proc. 10th int'l conf. cognitive modeling*, pages 241–246, 2010.

[2] Ransom K. Winder, James A. Reggia, Scott A. Weems, and Michael F. Bunting. An oscillatory hebbian network model of short-term memory. *Neural Computation*, 21:741–761, 2009.

[3] David Horn and Marius Usher. Parallel activation of memories in an oscillatory neural network. *Neural Computation*, 3:31–43, 1991.

[4] Vipin Srivastava, David Parker, and S.F. Edwards. The nervous system might 'orthogonalize' to discriminate. *Journal of theoretical biology*, 253:514–7, 09 2008.

[5] Sanjay G. Manohar, Yoni Pertzov, and Masud Husain. Short-term memory for spatial, sequential and duration information. *Current Opinion in Behavioral Sciences*, 17:20–26, 2017. Memory in time and space.

[6] Ramesh Naidu Annavarapu. On the memory storage capacity of hopfield networks using löwdin orthogonalizations. *International Journal of Advanced Scientific Technologies, Engineering and Management Sciences*, 2:18–23, 06 2016.

[7] Vipin Srivastava and S.F Edwards. A model of how the brain discriminates and categorises. *Physica A: Statistical Mechanics and its Applications*, 276(1):352–358, 2000.

[8] Vipin Srivastava, Suchitra Sampath, and David J. Parker. Overcoming catastrophic interference in connectionist networks using gram-schmidt orthogonalization. *PLOS ONE*, 9(9):1–7, 09 2014.

[9] Ralf Möller. First-order approximation of gram–schmidt orthonormalization beats deflation in coupled pca learning rules. *Neurocomputing*, 69(13):1582–1590, 2006. Blind Source Separation and Independent Component Analysis.

[10] Stanisław Jankowski, Zbigniew Szymański, Paweł Mazurek, and Jakub Wagner. Neural network classifier for fall detection improved by gram-schmidt variable selection. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 728–732, 2015.

[11] Toshifumi Minemoto, Teijiro Isokawa, Haruhiko Nishimura, and Nobuyuki Matsui. Pseudo-orthogonalization of memory patterns for complex-valued and quaternionic associative memories. *Journal of Artificial Intelligence and Soft Computing Research*, 7, 10 2017.

[12] Michael R. Davenport and Geoffrey W. Hoffmann. A recurrent neural network using tri-state hidden neurons to orthogonalize the memory space. *International Journal of Neural Systems*, 01(02):133–141, 1989.

[13] Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. In *International Conference on Learning Representations*, 2018.

[14] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. Information storage in neural networks with low levels of activity. *Phys. Rev. A*, 35:2293–2303, Mar 1987.

[15] K. Hornik and C.-M. Kuan. Convergence analysis of local feature extraction algorithms. *Neural Networks*, 5(2):229–240, 1992.

[16] R Eckhorn, R Bauer, W Jordan, M Brosch, W Kruse, M Munk, and H J Reitboeck. Coherent oscillations: a mechanism of feature linking in the visual cortex? multiple electrode and correlation analyses in the cat. *Biol. Cybern.*, 60(2):121–130, 1988.

[17] Terence D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2(6):459–473, 1989.

[18] Tao Zhu, Ye Xu, Furao Shen, and Jinxi Zhao. An online incremental orthogonal component analysis method for dimensionality reduction. *Neural Networks*, 85:33–50, 2017.

[19] Y.-F. Wang, J.B. Cruz, and J.H. Mulligan. Two coding strategies for bidirectional associative memory. *IEEE Transactions on Neural Networks*, 1(1):81–92, 1990.

[20] Kiyoshi Nishiyama and Haruhide Goto. An associate memory model based on hopfield neural network with redundant neurons. *Systems and Computers in Japan*, 24:96–107, 1993.

[21] Lurng-Kuo Liu and Panos A. Ligomenides. Unsupervised orthogonalization neural network for image compression. In David P. Casasent, editor, *Intelligent Robots and Computer Vision XI: Biological, Neural Net, and 3D Methods*, volume 1826, pages 215 – 225. International Society for Optics and Photonics, SPIE, 1992.

[22] Suchitra Sampath and Vipin Srivastava. On stability and associative recall of memories in attractor neural networks. *PloS one*, 15(9):e0238054, 2020.

[23] Hessameddin Akhlaghpour, Joost Wiskerke, Jung Yoon Choi, Joshua P Taliaferro, Jennifer Au, and Ilana B Witten. Dissociated sequential activity

and stimulus encoding in the dorsomedial striatum during spatial working memory. *Elife*, 5:e19507, 2016.